Objective-C & iOS Workshop
September 26th 2013

Patrick Jayet / @xrb
Oliver Gepp / @olivergepp
Sascha Thöni

iOS Workshop

13:00-14:00 Objective-C
- Presentation 30''
- Exercise 30''

14:00-15:30 iOS & UIKit
- Presentation 30''
- Break 30''
- Exercise 30''

11:30-12:30 UIKit Part 2
- Presentation 30''
- Exercise 30''

Objective-C Outline

- Classes & Interfaces
- Propertys
- Methods
- Categories
- Protocols
- Datatypes and Collection
- Utils

Objective-C overview

- object oriented programming language as superset from C
  - backward compatibility to C (it compiles any C program)
  - single inheritance like inJava
  - smalltalk-style messaging for calling methods
  - manages memory by reference counting
- invented in the early 1980s by Stepstone, licensed 1988 by NeXTstep and used 1996 by Apple Computer for Mac OS X
- main programming language for Mac OS X and iOS

Defining Classes & Interfaces

Person.**h**

```
#import <Foundation/Foundation.h>


@interface Person : NSObject
...
@end
```

Person.**m**

```

#import “Person.h”


@impementation Person : NSObject
...
@end
```

Propertys

- instance variables with generated accessor methods
- assignment behaviour (weak, strong, copy) could be controlled by attributes
- `@property` synthizes the accessor methods in the implementation
- `@synthesize` directive is no more needed since XCode 4.6

Person.h

```
@interface Person : NSObject

      @property NSString *firstName = nil;
      @property NSString *lastName;

@end
```

Property access

Person.**m**

```
#import “Person.h”

@impementation Person : NSObject

- (void) someMethod
{
     _firstName     = @”Regula”;                    //access variable directly
     self.firstName = @”Petra”;                     //calls setter method

     NSLog(@"firstname: %@", self.firstName);     //calls getter method
}
@end
```

Property attributes

Person.h

```
@interface Person : NSObject

     @property (readonly)    NSString *firstName;               //no setter generated
     @property (readwrite)   NSString *lastName;                //default


     @property (strong)  NSNumber *randomNumber;            //default
     @property (weak)        NSInteger *anotherNumber;
     @property (copy)        NSDouble *birthDate;               //needs NSCopying

     @property (atomic)  NSDate *dateFrom;                  //default
     @property (nonatomic)   NSDate *dateTo;                    //risk of corrupted data


@end
```

Methods .h

Person.h

```
#import <Foundation/Foundation.h>

@interface Person : NSObject

@property NSString *firstName;
@property NSString *lastName;

//initializer method
- (id) initWithName: (NSString*) aLastName andFirstName:(NSString*) aFirstName;

//class (factory) method
+ (Person*) personWithName: (NSString*) aLastName andFirstName:(NSString*) aFirstName;
@end
```

Methods .m

Person**.m**

```
#import "Person.h"
@implementation Person : NSObject

- (id) initWithLastName: (NSString*) aLastName andFirstName:(NSString*) aFirstName
{
    self = [super init];
    if (self) {
        _lastName = aLastName;
        _fistName = aFirstName;
    }
    return self;
}

+ (Person*) personWithLastName: (NSString*) aLastName andFirstName:(NSString*) aFirstName
{
     return [[self alloc] initWithLastName:aLastName andFirstName:aFirstName];
}

@end
```

Method invocation

AppController.**m**

```
#import “AppController.h”
#import "Person.h"

@implementation AppController : NSObject

- (id) init{
    self = [super init];
    if (self) {
        [self.person setFirstName:@"Homer"];            //message to nil return nil

        self.person =
                   [[Person alloc] initWithName:@"Simpson" andFirstName:@"Homer"];
        //or
        self.person = [Person personWithName:@"Simpson" andFirstName:@"Homer"];
    }
    return self;
}
@end
```

Private Properties & Methods

- There are no protected or private access modifiers for Objective-C methods, they are all public
- Private methods can be emulated by adding them to the implementation but not the interface.

Person.m

```
#import "Person.h"

@interface Person()

@property NSString* goodNightSong;
- (void) sleep;

@end

@implementation Person : NSObject
...
```

Protocols

“Defines a set of behavior that is expected of an object”

- are like interfaces in Java
- define a messaging contract
- supports declarations of instance methods, **class methods** and **properties**
- optional and required methods with directives
  - @required, @optional
- inherit from other protocols
- confirm to multiple protocols (comma seperated)

Protocols Example

EmployeeProtocol.**m**

```
#import "Job.h"
@protocol EmployeeProtocol <NSObject>
    @property Job *job;
    - (void) work;
    @optional
    - (void) getContract;
@end
```

Employee.**h**

```
#import "EmployeeProtocol.h"
@interface Employee : NSObject <EmployeeProtocol>
@end
```

Employee.**m**

```
#import "Employee.h"
@implementation Employee
@synthesize job;
- (void)work{
}
@end
```

Datatypes & Collections

Basic C

- Primitive datatypes
  - int, double, float, char…
- Operators: ++, --

Objectiv-C

- Primitives datatypes
  - BOOL, SEL, id, Class
- Common Types
  - NSObject, NSNumber, NSString, NSURL
- Collections
  - NSArray, NSDictionary, NSSet

Collections - Example

```
...
NSArray *carsFromObjects = [NSArray arrayWithObjects:@"VW", @"BMW", @"Porsche", @"VW", nil];
NSLog(@"%@", carArray[0]);
NSLog(@"%@", [carArray objectAtIndex:0]);


...
NSSet *carSet = [NSSet setWithObjects:@"VW", @"BWM", @"Porsche", nil];
carSet = [NSSet setWithArray:self.carList];                      //make array elements unique


for (id item in carSet) {                                        //fast enum
     NSLog(@"%@", item);
}


...
NSDictionary *carDict = [NSDictionary dictionaryWithObjectsAndKeys: @1, @"VW", @5, @"BWM", @45,
@"Porsche", nil];


NSLog(@"There are %@ BMW's in stock", carDict[@"BMW"]);
NSLog(@"There are %@ VW's in stock", [carDict objectForKey:@"VW"]);
...
```

Customizing Existing Classes with Categories

“With categories and class extensions you can add new behaviour to existing classes”

by categories:

- works only for (instance/class) methods, property wouldn’t be synthesized
- any methods are available to all subclasses
- you can add to every class, framework even if only compiled class is available
- className convention: `className+categoryName.h`

Syntax

```
@interface ClassName (CategoryName)


@end
```

Customizing Existing Classes with Categories - Example

Person+PersonExtended.h

```
@interface Person  (PersonExtended)
      - (NSString*) fullName;
@end
```

Person+PersonExtended.m

```
#import "Person+PersonExtended.h"
@implementation Person  (PersonExtended)

- (NSString *)fullName {
      NSMutableString* fullName = [NSMutableString stringWithString:self.firstName];
      [fullName appendString:@" "];
      [fullName appendString:self.lastName];
      return fullName;
}
@end
```

AppController.h

```
#import "Person+PersonExtended.h"
...
      NSLog(@"Name: %@", [self.person fullName]);              //Console -> “Name: Homer Simpson”
...
```

Utils

Override object description

- it’s like `toString()` in Java

```
- (NSString *)description
{
    return [self.firstName stringByAppendingString:self.lastName];
}
```

NSLog

- supports format characters from C printf()
- %@ for writing string or object description

```
...
NSLog(@"string: %@, float: %f, double %f, int: %d", @"Text", 3.141592654f, 3.5, 5);
...
```

Exercise 1

- Create a project
- Run a first test

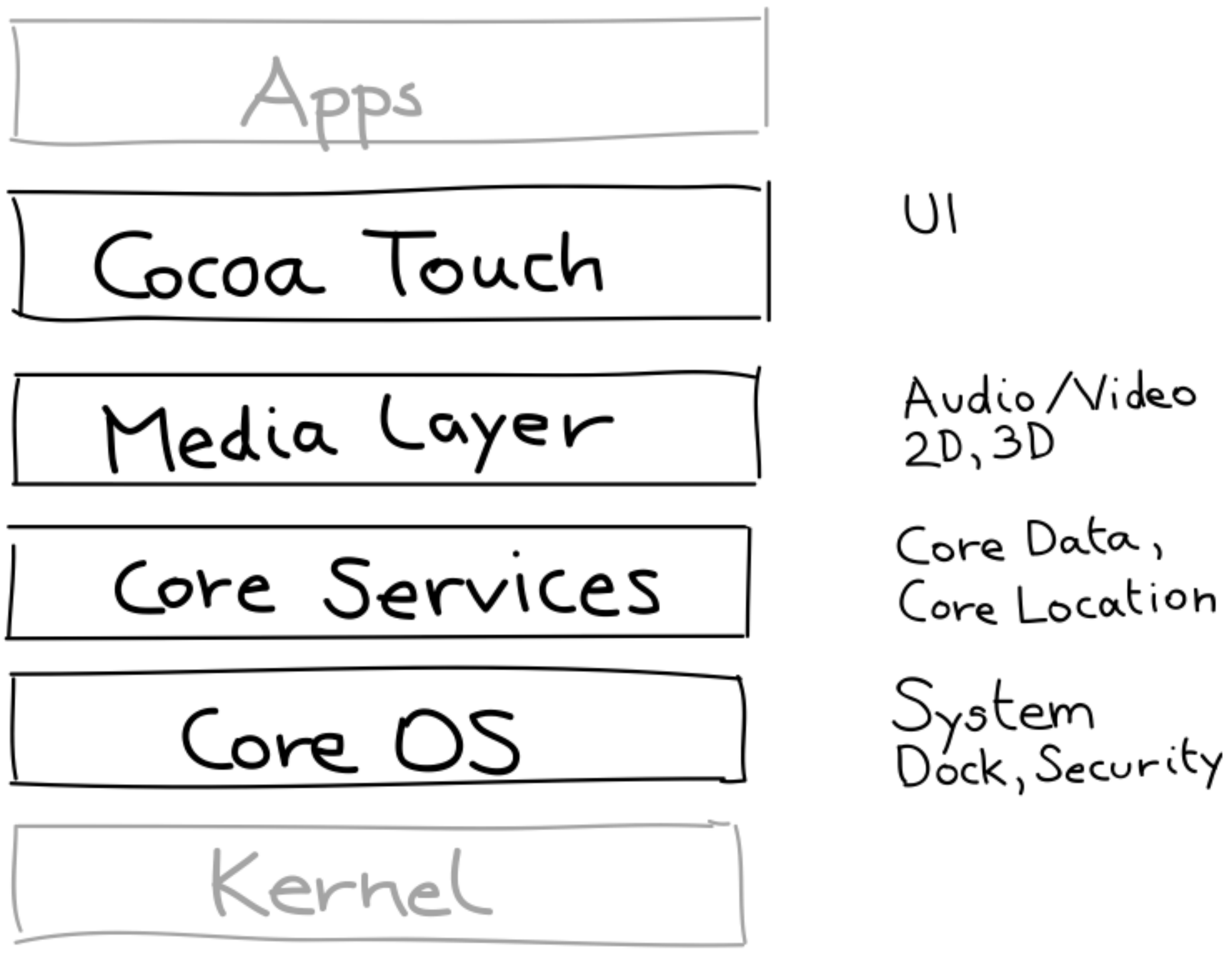
Download Exercise from:

https://github.com/pajai/RpnCalc

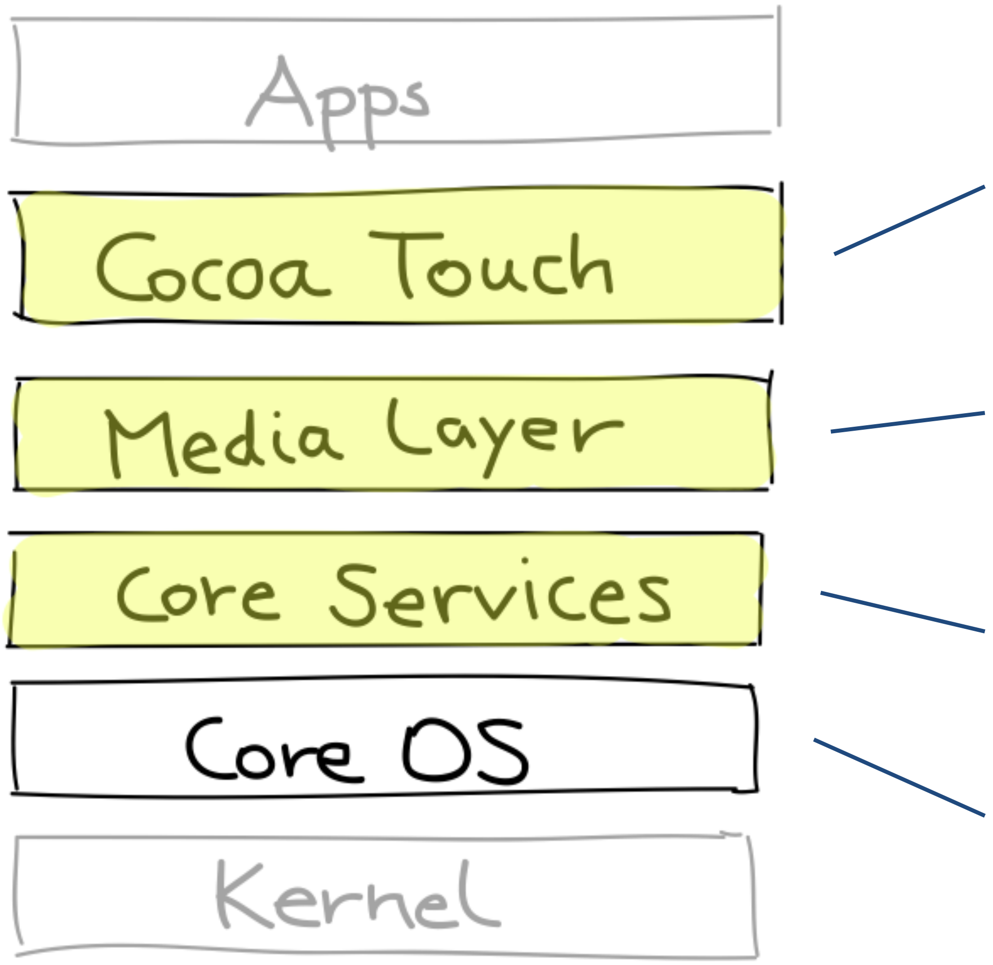
iOS & UIKit Outline

- iOS
    - Overview
- App lifecycle
- View hierarchy
- View controller lifecycle
- MVC
- Interface Builder
- Exercise

iOS – Layers


Apps
Cocoa Touch
Media Layer
Core Services
Core OS
Kernel
UI
Audio/Video
2D, 3D
Core Data,
Core Location
System
Dock, Security

iOS Frameworks



UIKit, MapKit,
Standard System View Controllers
(Contact, E-Mail, Calendar, …), etc.

CoreGraphics, CoreAnimation,
OpenGL, AVFoundation, etc.

CoreData, CoreLocation, EventKit,
Foundation, etc.

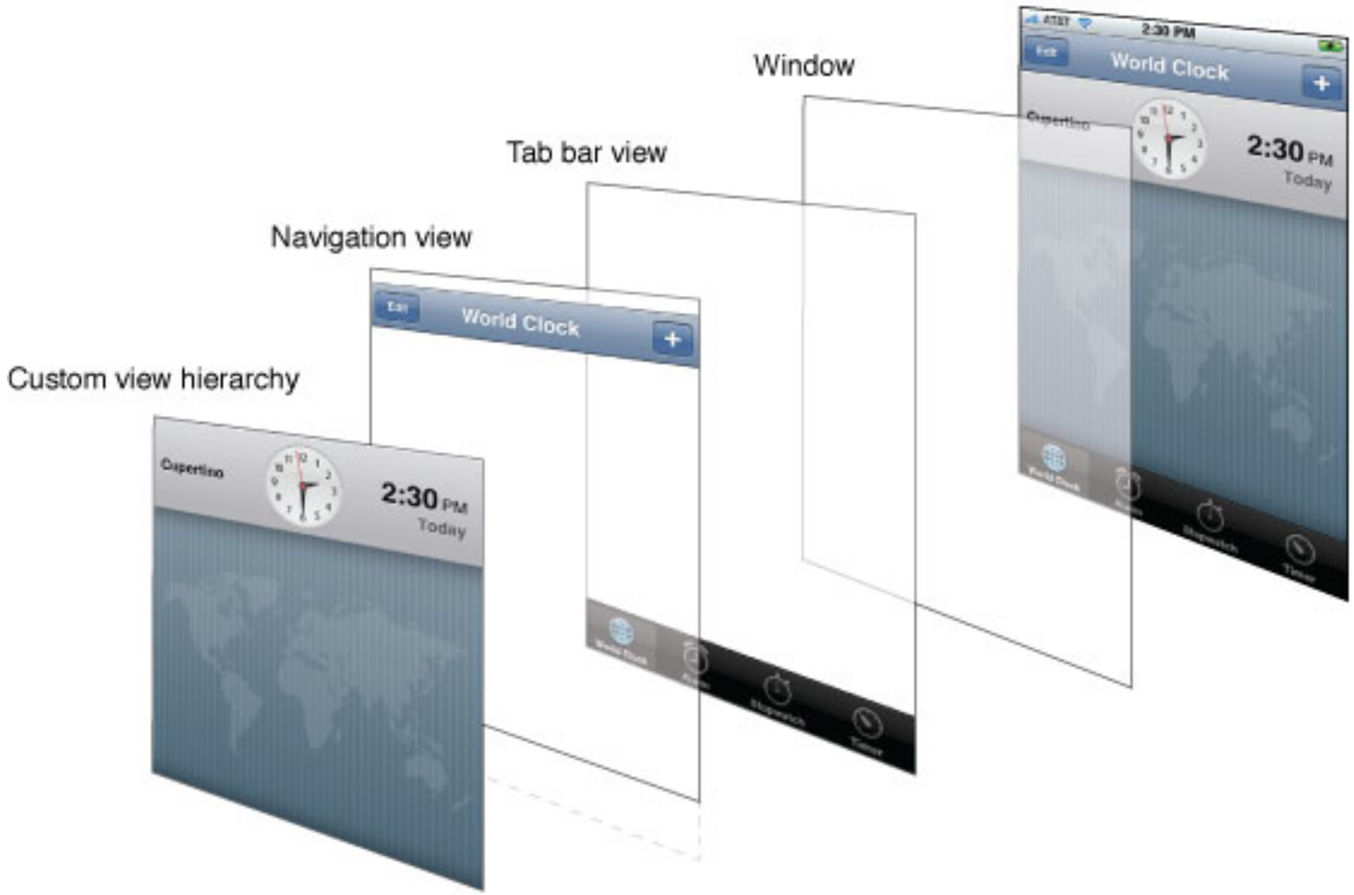
CoreBluetooth, External Accessory,
Security Framework

App Lifecycle

- Each app has exactly one App Delegate
- App Delegate receives notifications
    - Launch terminated
    - App will terminate
    - App goes to background / comes to foreground
    - …

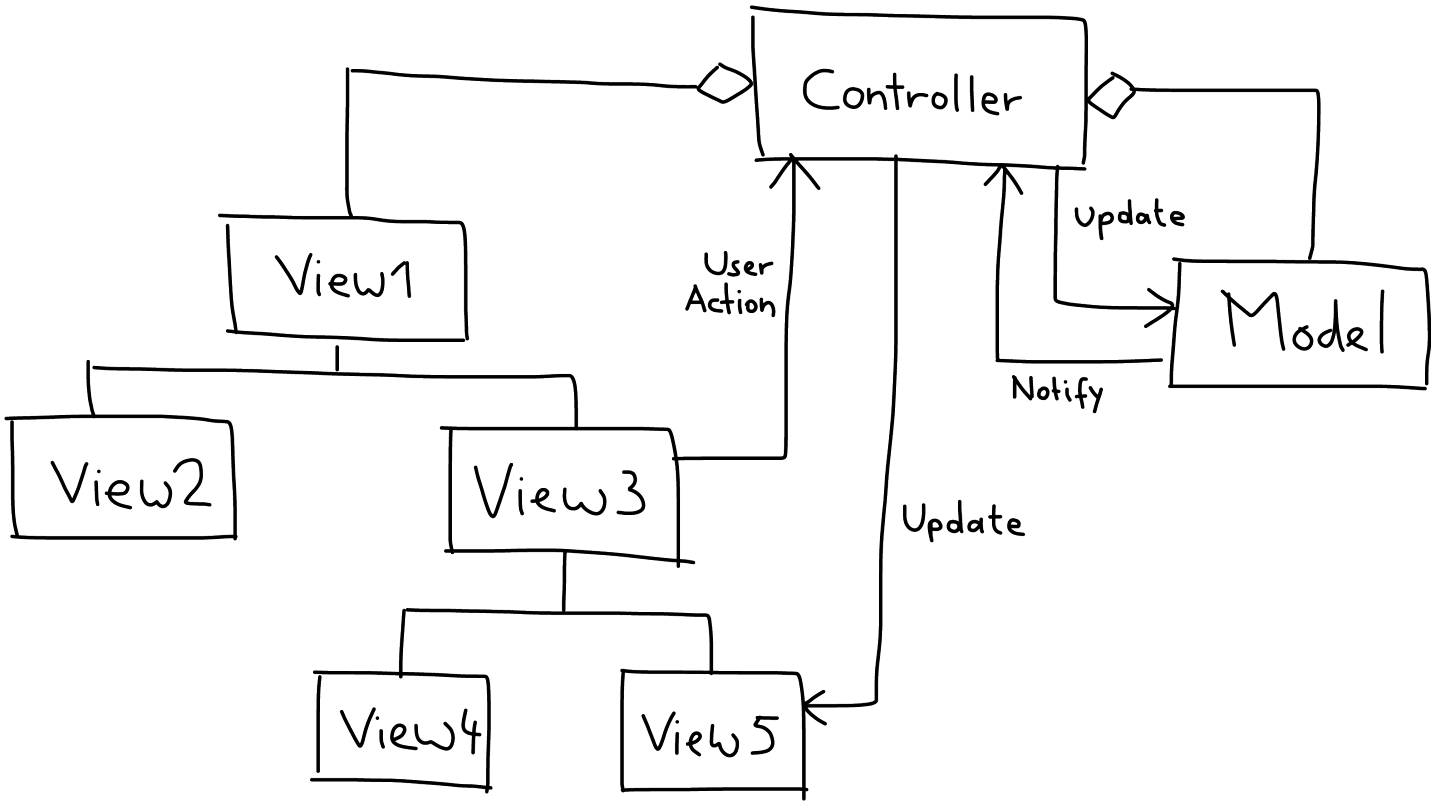
→ Show in Xcode

View Hierarchy



→ Show in Xcode

MVC


Controller
View1
View2
View3
View4
View5
Model
User Action
Update
Update
Notify

View Controller Lifecycle

- Receives notifications for its main view
  - Loaded
  - Appeared
  - Disappeared
  - …

→ Show in Xcode

Interface Builder

- NextStep (1986)
- Since Xcode 4: part of the IDE
- Screen & storyboards

Interface Builder

- NextStep (1986)
- Since Xcode 4: part of the IDE
- Screen & storyboards

- Link items in IB with code
  - Class
  - Outlet of a view
  - Callback method (user event)

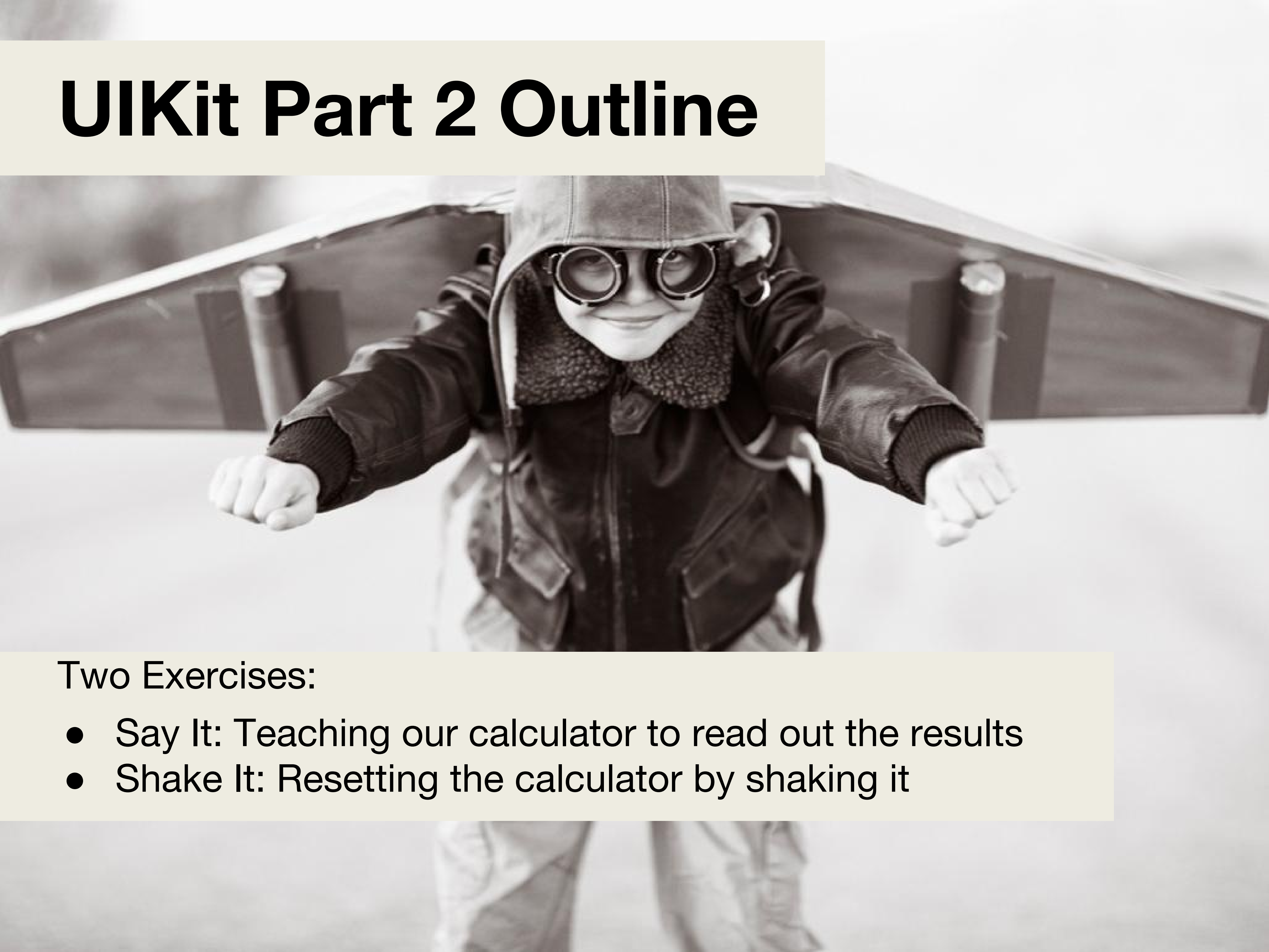
→ Show in Xcode

Break

- 14:30 – 15:00

iOS & UIKit

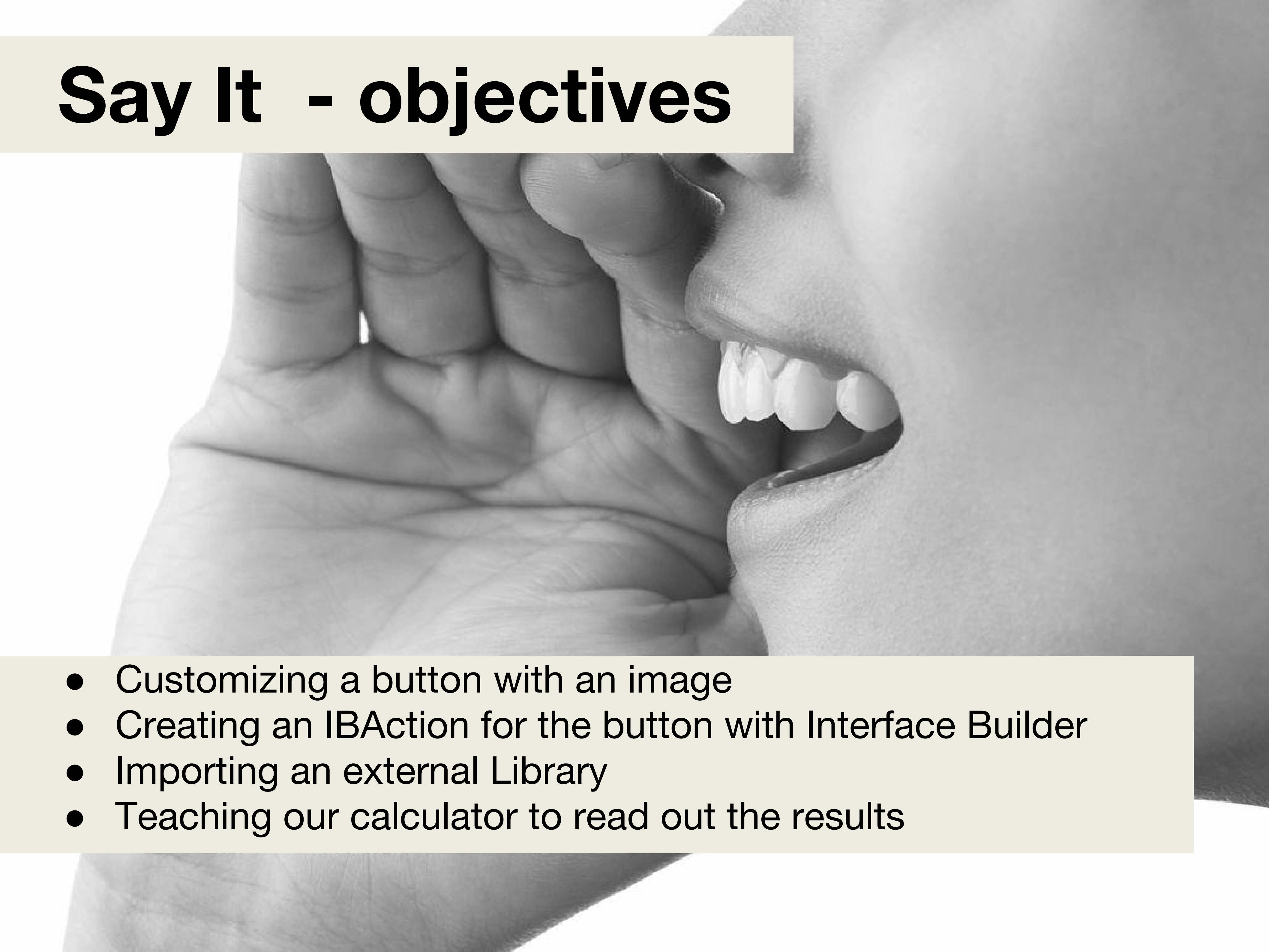
- Exercise
  - Build an RPN Calculator



UIKit Part 2 Outline

Two Exercises:

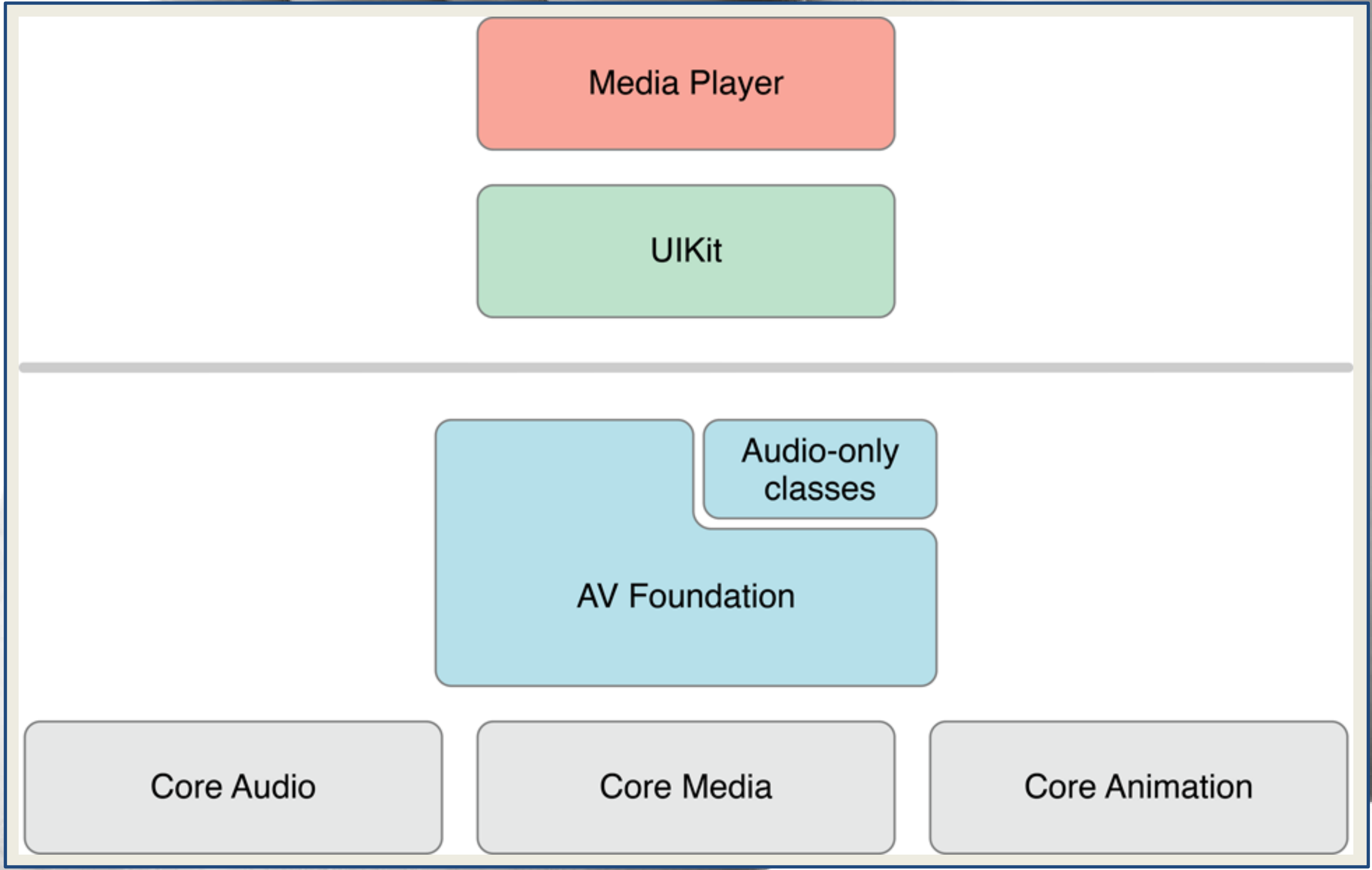
- Say It: Teaching our calculator to read out the results
- Shake It: Resetting the calculator by shaking it



Say It - objectives

- Customizing a button with an image
- Creating an IBAction for the button with Interface Builder
- Importing an external Library
- Teaching our calculator to read out the results

AVFoundation



Say It

Demo

Shake It - objectives

- Adding some code to react on shake events
- Simulating shake events with the iPhone simulator
- Modifying the model to reset the calculator

Shake It

Demo

Thank You!

- Questions?

References

- App iOS Programming Guide http://goo.gl/wzyMTQ
- Xcode Overview http://goo.gl/ptZQGK
- UIKit User Interface Catalog http://goo.gl/5Bkf6V